Kelly Hancox
CIS 678 Machine Learning
Decision Tree

Implementation:

Reading in data

I read in the data using Python's open and read methods. I then split the data along the new lines and put the data into a dictionary with attributes, classes, and s. I got this idea from my friend Allison Bolen who took this class last year. Below is an example of this dictionary:

```
{   'attributes': {   'age': {   'index': 0,
                                 'num': '3',
                                 'values': [   'young',
                                               'pre-presbyopic',
                                               'presbyopic']},
                      'astigmatism': {   'index': 2,
                                         'num': '2',
                                         'values': ['no', 'yes']},
                      'num': 4,
                      'prescription': {   'index': 1,
                                          'num': '2',
                                          'values': ['myope', 'hypermetrope']},
                      'tear-rate': {   'index': 3,
                                       'num': '2',
                                       'values': ['reduced', 'normal']}},
    'classes': {'num': 3, 'values': ['soft', 'hard', 'none']},
    's': [  #examples here ],
    'total': 24}
```

Entropy

Next, I find the overall entropy using a helper method that takes the sample data, creates a dictionary that counts the number of times a value appears and calculates the entropy based off of these values and the total number of samples passed to it.

Id3

Then, id3 is called with 's', 'attributes', and 'classes' from the sample dictionary. Id3 finds the current classes from within the samples from a helper method and saves those into another dictionary. It checks if there is only one class and if there is, then it returns with the value from that class. Next, it checks if there are no attributes remaining and then returns the most common class with a helper method. If neither of these are true, id3 finds the attribute with the most gain through two helper methods.

Gain

The first gain helper method is given s, the attributes, and the classes, and it creates a dictionary of gains for all of the attributes, then returns the name of the attribute with the highest gain. It finds the gain by passing another method the values of an attribute, for instance ‘Sunny’, ‘Cloudy’, ‘Rainy’, the samples, the index of the attribute in the samples, and the classes. This second gain method creates a double array of counts with size [[classes] x [variables]]. Then it sums the numbers of those arrays and returns the gain to the first function, which collects the gain of all of the attributes to return the name of the attribute with the highest gain.

Recursion

Once id3 has the attribute with the most gain, it uses that string value to parse through the dictionary of attributes and find its values. For instance, the values of [‘Sunny’, ‘Cloudy’, ‘Rainy’] for ‘Forecast’ are found. Then the method searches through the samples and creates an array of samples for each of the values of that attribute. For instance, Sunny, Cloudy, and Rainy would all have their own arrays saved in a dictionary with those values as the keys. Id3 deletes the most common attribute from the attribute array, and uses a for loop through the items in that dictionary to then recursively call id3 with each of those samples. Finally, it returns a dictionary with these values.

Experiments Performed:

Training and testing

To test the effectiveness of this tree, the sample data is split into test and training data. I shuffle the instances, calculate the number of instances for 80% of the data, which I used for training, and use the other 20% for testing. I run the id3 with the training data and then loop through the test data and send it to another function that I called classify_me. This function looks down the levels of the keys in the final dictionary and gets down to a classification for a particular example. If this example matches the correct classification, then I add it to a correct_classifications count. If it does not, then I add it to an incorrect_classification count.

With the small sample size of 24 for contact lenses, the classifier did not make sense. It had at best a 100% accuracy rate and at worst a 20% accuracy rate. I then tried the same classifier with car data that I found from the internet and with an 80:20 ratio of training to test data. I ran this 10 times and at most, my classifications were 89.6% and at worst 85.8% accurate.

Example output:
Correct classifications 310
Incorrect classifications 36
89.6% correct

Example output:
Correct classifications 297
Incorrect classifications 49
85.8% correct

Visualizations

Pretty Printer

One way that I printed the final result was using pretty print to print out the dictionary in a more readable format. I import pprint and define pretty print to use an indent of 4. Below is an example of the print out.

```
{   'tear-rate': {   'normal': {   'astigmatism': {   'no': {   'age': {   'pre-presbyopic': 'soft',
                                                                             'presbyopic': {   'prescription': {   'hypermetrope': 'soft',
                                                                                                                   'myope': 'none'}},
                                                                             'young': 'soft'}},
                                                    'yes': {   'prescription': {   'hypermetrope': {   'age': {   'pre-presbyopic': 'none',
                                                                                                                   'presbyopic': 'none',
                                                                                                                   'young': 'hard'}},
                                                                                   'myope': 'hard'}}}},
                     'reduced': 'none'}}
```

Printing the path of the dictionary

Another example of printing out the result happened because I tried to make a function out of the words in the dictionary for the test data. This was not the method I ultimately used for the test data, but it made the results into readable sentences, like the ones below:

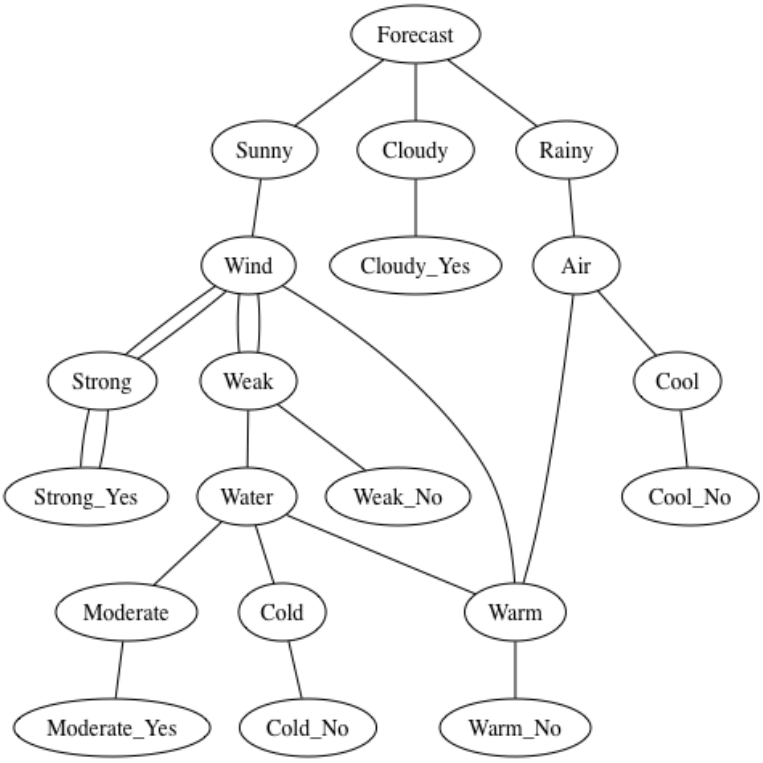
```
(base) toto:DecisionTree kellyhancox$ python3 decisionTree.py
if Forecast is Sunny and Wind is Strong, then recommendation is: Yes
if Forecast is Sunny and Wind is Weak and Water is Warm, then recommendation is: No
if Forecast is Sunny and Wind is Weak and Water is Moderate, then recommendation is: Yes
if Forecast is Sunny and Wind is Weak and Water is Cold, then recommendation is: No
if Forecast is Cloudy, then recommendation is: Yes
if Forecast is Rainy and Air is Warm and Wind is Strong, then recommendation is: Yes
if Forecast is Rainy and Air is Warm and Wind is Weak, then recommendation is: No
if Forecast is Rainy and Air is Cool, then recommendation is: No
```

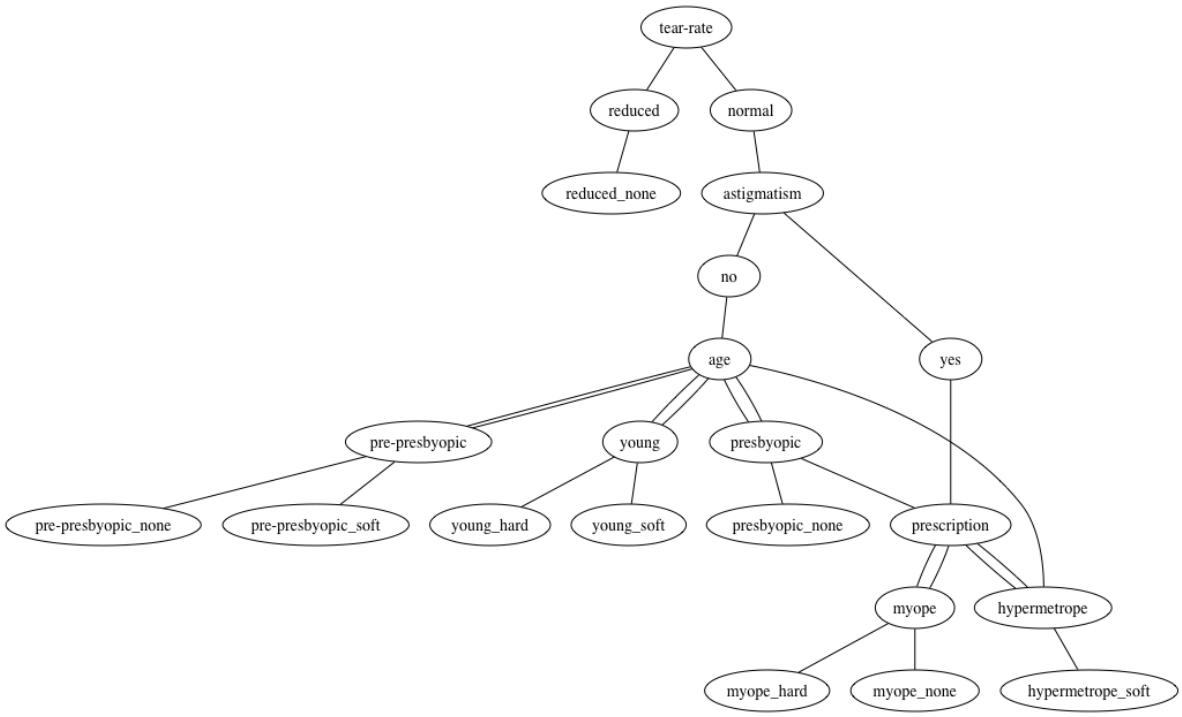
```
(base) toto:DecisionTree kellyhancox$ python3 decisionTree.py
if tear-rate is reduced, then recommendation is: none
if tear-rate is normal and astigmatism is no and age is young, then recommendation is: soft
if tear-rate is normal and astigmatism is no and age is pre-presbyopic, then recommendation is: soft
if tear-rate is normal and astigmatism is no and age is presbyopic and prescription is myope, then recommendation is: none
if tear-rate is normal and astigmatism is no and age is presbyopic and prescription is hypermetrope, then recommendation is: soft
if tear-rate is normal and astigmatism is yes and prescription is myope, then recommendation is: hard
if tear-rate is normal and astigmatism is yes and prescription is hypermetrope and age is young, then recommendation is: hard
if tear-rate is normal and astigmatism is yes and prescription is hypermetrope and age is pre-presbyopic, then recommendation is: none
if tear-rate is normal and astigmatism is yes and prescription is hypermetrope and age is presbyopic, then recommendation is: none
```

Pyplot and graphviz

My final representation of the final result is a graph created with pydot and graphviz. It uses two methods, one that finds nodes of the dictionary and uses the draw method that uses pydot and graph to draw the figure. Below are the figures for contact lenses and fishing data.

They are helpful to understand the first few levels, but not entirely usable because each node is reused unless it is a leaf node, so it is impossible to tell which direction the line is going. It is more helpful with small amounts of data, as the fishing data graph is even slightly more readable than the contact lense data graph.





Problems encountered:

Data saving

I had to think a lot about how to save all of the data. I asked friends how they stored the data and they suggested a dictionary, so I took a hand at doing that throughout the process. It was pretty effective. I think the double array that I use in the get_gain method is inefficient, but when I was looking at the worksheet, I couldn't keep thinking about a double array, so I used it.

Recursion

I had difficulty recursively calling the id3 method. I was trying to pass the attribute dictionary in, but when it was called on the new attribute, then it was still trying to use the attribute dictionary that had already deleted other attributes. Even after creating a shallow copy, it still encountered the same problem. I had to create a deep copy of the attribute dictionary to solve the problem.

Figuring out how to return the results was also challenging. I considered passing an empty dictionary into the method and adding to it throughout the calls, but decided to just have a dictionary created within the first instance and to have everything returned to that instance.

Another mistake I made was that I was accidentally deleting one sample every time I came upon an attribute that hadn't been seen yet. I was making the array to place the sample inside, but was not adding the sample to the array. I used print statements of 's' to find this bug.

Graphviz

When I first installed pydot and graphviz, I had to use ‘brew install graphviz’, which inadvertently modified my path and I had to reinstall python3 and a few other libraries. This is why I was unable to work more with graphviz to get a more accurate graph of the data.

Code:

```
import re
import string
import matplotlib.pyplot as plt
import numpy as np
import math
import copy
import random
import pprint
import pydot
pp = pprint.PrettyPrinter(indent = 4)



#source:
https://stackoverflow.com/questions/13688410/dictionary-object-to-decision-tree-in-pyd
ot
def draw(parent_name, child_name):
   edge = pydot.Edge(parent_name, child_name)
   graph.add_edge(edge)

def visit(node, parent=None):
   for k,v in node.items():
       if isinstance(v, dict):
           # We start with the root node whose parent is None
           # we don't want to graph the None node
           if parent:
               draw(parent, k)
           visit(v, k)
       else:
           draw(parent, k)
           # drawing the label using a distinct name
           draw(k, k+'_'+v)
```

```
#source
https://stackoverflow.com/questions/34836777/print-complete-key-path-for-all-the-value
s-of-a-python-nested-dictionary
def dict_path(path,my_dict):

    for k,v in my_dict.items():
        if isinstance(v,dict):
            dict_path(path+" "+k,v)
        else:
            line = path+ " " +k
            line = re.split(r" ", line)
            line = line[1:]
            fullLine = "if "

            for index in range(0, len(line)-1, 2):
                if index == len(line)-2:
                    fullLine = fullLine + line[index] + " is " + line[index+1]

                else:
                    fullLine = fullLine + line[index] + " is " + line[index+1] + " and
"

            fullLine = fullLine + ", then recommendation is: " + v
            print(fullLine)


def id3(s, attributes, classes):

    current_classes = return_classes(s)

    if len(s) == 0:
        return

    #if all examples in s are of the same class
    if len(current_classes.keys()) == 1:
        for key in current_classes.keys():
            return key
```

```
    #else if no attributes left
    elif attributes['num'] == 0:
        return get_most_common_class(current_classes)

    #else choose the attribute that maximizes the gain
    else:
        attribute_with_most_gain = get_most_gain(s, attributes, classes)
        currentDict = {}
        for instance in s:
            instanceIndexed = re.split(r",", instance)

            for value in attributes[attribute_with_most_gain]['values']:

                if value not in currentDict:
                    currentDict[value] = []
                if instanceIndexed[attributes[attribute_with_most_gain]['index']] ==
value:
                    currentDict[value].append(instance)


        del attributes[attribute_with_most_gain]
        attributes["num"] = attributes['num'] - 1

        for key, value in currentDict.items():
            attributes2 = copy.deepcopy(attributes)
            currentDict[key] = id3(value, attributes2, classes)


        finalDict = {attribute_with_most_gain: currentDict}
        return finalDict


def get_most_gain(s, attributes, classes):

    keys = attributes.keys()
    gains = {}
    for key in keys:
        if key != 'num':
```

```
            gains[key] = get_gain(attributes[key]['values'], s,
attributes[key]['index'], classes)

    #return the maximum of the gains
    return max(gains, key=gains.get)



def get_gain(variables, instances, index, classes):

    temp = [ [ 0 for y in range( len(classes['values']) ) ] for x in range(
len(variables) ) ]

    gain = []
    for instance in instances:
        instance = re.split(r",", instance)

        for j, variable in enumerate(variables):
            #index is the position of the attribute
            if instance[index] == variable:

                for k, class_a in enumerate(classes['values']):
                    if instance[len(instance)-1] == class_a:
                        if temp[j][k] == 0:
                            temp[j][k] = 1

                        else:
                            temp[j][k] = temp[j][k] + 1


    for i, line in enumerate(temp):
        summation = 0
        total = 0

        for val in line:

            if val > 0 and sum(line) > 0:
                summation = summation + val/sum(line)*math.log2(val/sum(line))
                total = total + val
```

```
            else:
                summation = summation + 0
                total = total + val

        currEntropy = summation

        if len(instances)*summation != 0:
            entropyForFormula = total/len(instances)*summation
        else:
            entropyForFormula = 0

        gain.append(entropyForFormula)

    finalGain = entropy + sum(gain)
    return finalGain


def get_most_common_class(current_classes):
    return max(current_classes, key=current_classes.get)


def return_classes(s):
    current_classes = {}

    for example in s:
        example = re.split(r",", example)
        if example[len(example)-1] in current_classes:
            current_classes[example[len(example)-1]] =
current_classes[example[len(example)-1]] + 1
        else:
            current_classes[example[len(example)-1]] = 1

    return current_classes



def get_entropy(s):

    countingDict = {}
```

```
    total = 0
    for instance in s:

        instance = re.split(r",", instance)
        total = total + 1

        if instance[len(instance)-1] in countingDict:
            countingDict[instance[len(instance)-1]] =
countingDict[instance[len(instance)-1]] + 1
        else:
           countingDict[instance[len(instance)-1]] = 1

    entropyList = []
    for val in countingDict.values():
        value = val/total*math.log2(val/total)
        entropyList.append(value)

    finalEntropy = -sum(entropyList)
    return finalEntropy


def classify_me(example, dictionary, index_labels):

    classification = ""

    for key in dictionary.keys():
        currKey = key
        index = index_labels[key]
        currentVariable = example[index]

        for key2 in dictionary[currKey].keys():

            if currentVariable == key2:
                if isinstance(dictionary[currKey][key2],dict):
                    classification = classify_me(example, dictionary[currKey][key2],
index_labels)
                else:
                    return dictionary[currKey][key2]
```

```
    return classification

def get_index_labels(attributes):

    index_labels = {}
    for key in attributes.keys():
        if key != 'num':
            index_labels[key] = attributes[key]['index']

    return index_labels

def is_an_int(s):
    try:
        int(s)
        return True
    except ValueError:
        return False

#read in contents
file = open("fishingData.txt", "r")
contents = file.read()
info = re.split(r"\n", contents)

#create dictionary with information
fileDictionary = {"s": []}
count = 0
for index in range(0, len(info)):

    if is_an_int(info[index]) and count == 0:
        # class values
        fileDictionary["classes"] = {"num": int(info[index]), "values":
info[1].split(",")}
        count = count + 1

    elif is_an_int(info[index]) and count == 1:
        fileDictionary["attributes"] = {"num":int(info[index])}
        count = count + 1

        for attributeIndex in range(index+1, index+1+int(info[index])):
```

```
            attribute = info[attributeIndex].split(",")[0]
            numValues = info[attributeIndex].split(",")[1]
            values = info[attributeIndex].split(",")[2:]
            fileDictionary["attributes"][attribute] = {"index": attributeIndex-3,
"num":numValues, "values":values}

    elif is_an_int(info[index]) and count == 2:
        fileDictionary["total"] = int(info[index])
        count = count + 1

    elif count == 3:
        fileDictionary["s"].append(info[index])



entropy = get_entropy(fileDictionary['s'])
sArray = fileDictionary['s']
copyOfInfo = copy.deepcopy(fileDictionary)

random.shuffle(sArray)
amountOfTraining = round(len(sArray)*.8)
training = sArray[0:amountOfTraining]
test = sArray[amountOfTraining:]

final = id3(fileDictionary['s'], fileDictionary["attributes"],
fileDictionary["classes"])
index_labels = get_index_labels(copyOfInfo["attributes"])

#pp.pprint(final)
#dict_path("", final)

correct_classification = 0
incorrect_classification = 0

for example in test:
    example = re.split(r",", example)
    classification = classify_me(example, final, index_labels)
    if classification == example[len(example)-1]:
        correct_classification = correct_classification + 1
    else:
```

```
        incorrect_classification = incorrect_classification + 1

#print("Correct classifications", correct_classification)
#print("Incorrect classifications", incorrect_classification)



#graphing with pydot
graph = pydot.Dot(graph_type='graph')
visit(final)
graph.write_png('fishing_graph2.png')
```